LaniEngine - Data Object Model

The **DOM** or **Data Object Model** (not to be confused with the GOM) is a collection of dynamic definition objects. The DOM is where node classes are created and defined, fields in those classes are further specified, certain field values are enumerated, and prototypes are created. GOM data is then placed in those fields.

- Changes to DOM structure can be modified by the CLI or the DOM Editor
- GOM data, on the other hand, can be modified from the CLI, or also from the LSL scripting language.
- DOM structures cannot be modified from LSL.
- Different versions of the DOM exist on the client and server, each of which is stored in a separate database schema, for client or server, as appropriate.

Structure of a node

- A GOM node is always created based on a class, a prototype, or another node.
- Each node is defined by one or more classes, each of which has been based on a particular Archetype.
- A node can have multiple classes.
- A node's fields are directly dependent on which classes it has
- Each field can contain different types of data, depending on the field datatype.
- Fields are associated with classes. A node cannot have a field, unless that field is part of a class which is on the node. Once a class is attached to a node, that node then has access to all the fields of that class.
- If multiple classes on a node duplicate the same field, the node only has access to one field of that type -- it doesn't have duplicate fields.

Prototypes

Prototypes are a combination of DOM *structure* and GOM *data*. For more information, see the section on Prototypes.

DOM Reflection via LaniScript Language

Definitions stored in the DOM are exposed to LSL via a series of external functions, using those functions it is possible to query the DOM for information on classes, including their inheritance hierarchy and listing their member fields.

Get Parent Classes of a Class

The external function **GetParentClasses()** retrieves the immediate parents of a child class.

```
ParentsOfClass as List of String = GetParentClasses( className )
```

List Member Fields of a Class

A class is comprised of its own member fields and any fields it inherits from its parent classes. The external function **GetClassFields()** retrieves the member fields of a class, but not any fields it inherits.

```
fields as list of string = GetClassFields( className )
```

Retrieving all of the fields that class has or inherits requires you get the parent classes, get their fields, and recurse until you reach the penultimate parent class(es) removing any duplicates along the way.

```
public function getAllFieldsInClass( className as String ) as List of String
  classes as List of String
  // call a function to get the parent classes and all of their parent classes
recursively

  getAllParentsForClass( className, classes )
  fields as List of String = getClassFields( className )

  foreach c in classes
    classFields as List of String = GetClassFields( c )

    alreadyInList as Boolean

    // eliminate duplicate adds
    foreach cf in classFields
      foreach f in fields
        if f = cf
          alreadyInList = true
          break
        .
      .
      if not alreadyInList
        add back cf to fields
      .
    .
  .

  return fields
.

public function getAllParentsForClass(Class as String, ClassList references
List of String)
  ParentsOfClass as List of String = GetParentClasses(Class)


  // For all of the parents, check for whether or not they are already in the
class list
  //    if not, add them to the list and then recurse to walk their parent
classes
  foreach p in ParentsOfClass
```

```
    AlreadyInClassList as Boolean

    foreach c in ClassList
      if (c == p)
        AlreadyInClassList = true

        break
      .
    .

    if not AlreadyInClassList
      add back p to classList

      // recurse
      getAllParentsForClass(p, ClassList)
    .
  .
.
```

DOM Coordination

DOM Coordination is a process whereby the DOM Coordinator process notifies all running server processes that they must come to a stable state (i.e. essentially stop executing any code including LaniScript) to apply a modification to the DOM as one whole atomic unit. A Coordinated DOM change is applied to all server processes simultaneously and if any process fails to apply the change it is rolled back for all processes.

It is acceptable and expected for coordinated changes to be made (constantly) during development, however in a production (live) world running hundreds or thousands of server processes getting all of the processes to come to that stable state **may** take an unacceptably long period of time. Consequently, any changes that may result in a coordinated DOM change should be rare in your production world(s).

Coordinated changes

- Modifications to the Server DOM
- Create/Delete/Updates to Server Prototypes
- (some) Updates via Live Update

Parameters

Note: Only customers with full or source licenses can modify values via the Master Control Console.

Parm Name	Parm Type	Description	Default value

JavelinConnectString	string	complete schema password database and identifier	javelin/javelin@somewhere
World	string	prefix used on well-known mailbox names	0
HEBatchIDLine	string	selects the line of IDs to draw from when requesting IDs from the ID Server	HE
BatchIDClientTimeout	integer	how long in milliseconds to wait for a response from the ID Server	7000
ServerHeartbeatInterval	integer	frequency in milliseconds at which to query coordinated servers to make sure that they are still there	120000
CoordinationTimeout	integer	milliseconds to wait for all servers to validate a DOM change before considering it timed out	1000

DOM Editor

The **DOM Editor** is a tool for viewing and editing DOM definitions visually as an alternative to using the CLI directly. The information on this page relates to the DOM Editor 2.0, which improves functionality and usability, and separates viewing and editing of the client and server DOMs into independent windows.

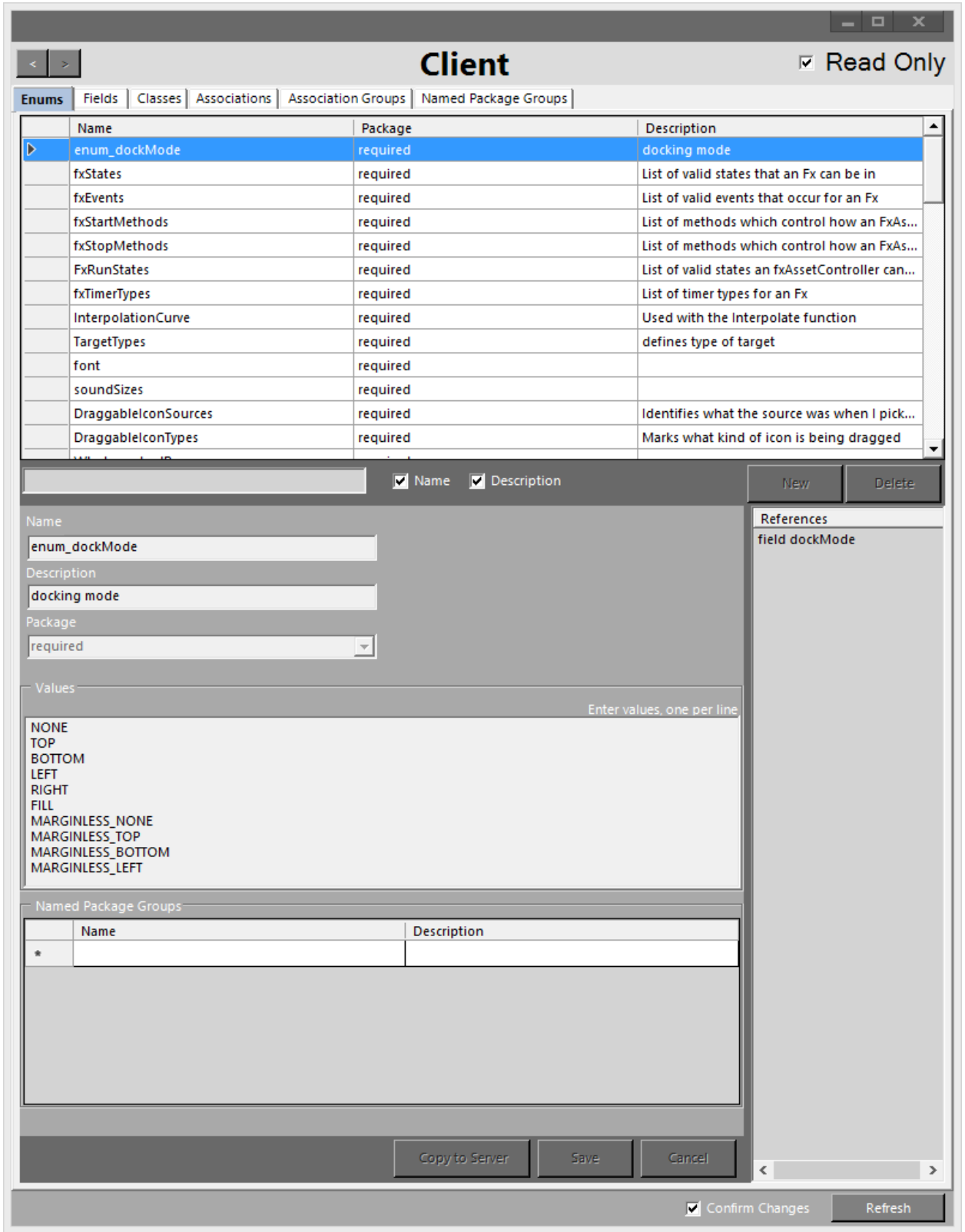
NOTE: For information on the previous version of the DOM Editor, please refer to DOM Editor 1.0

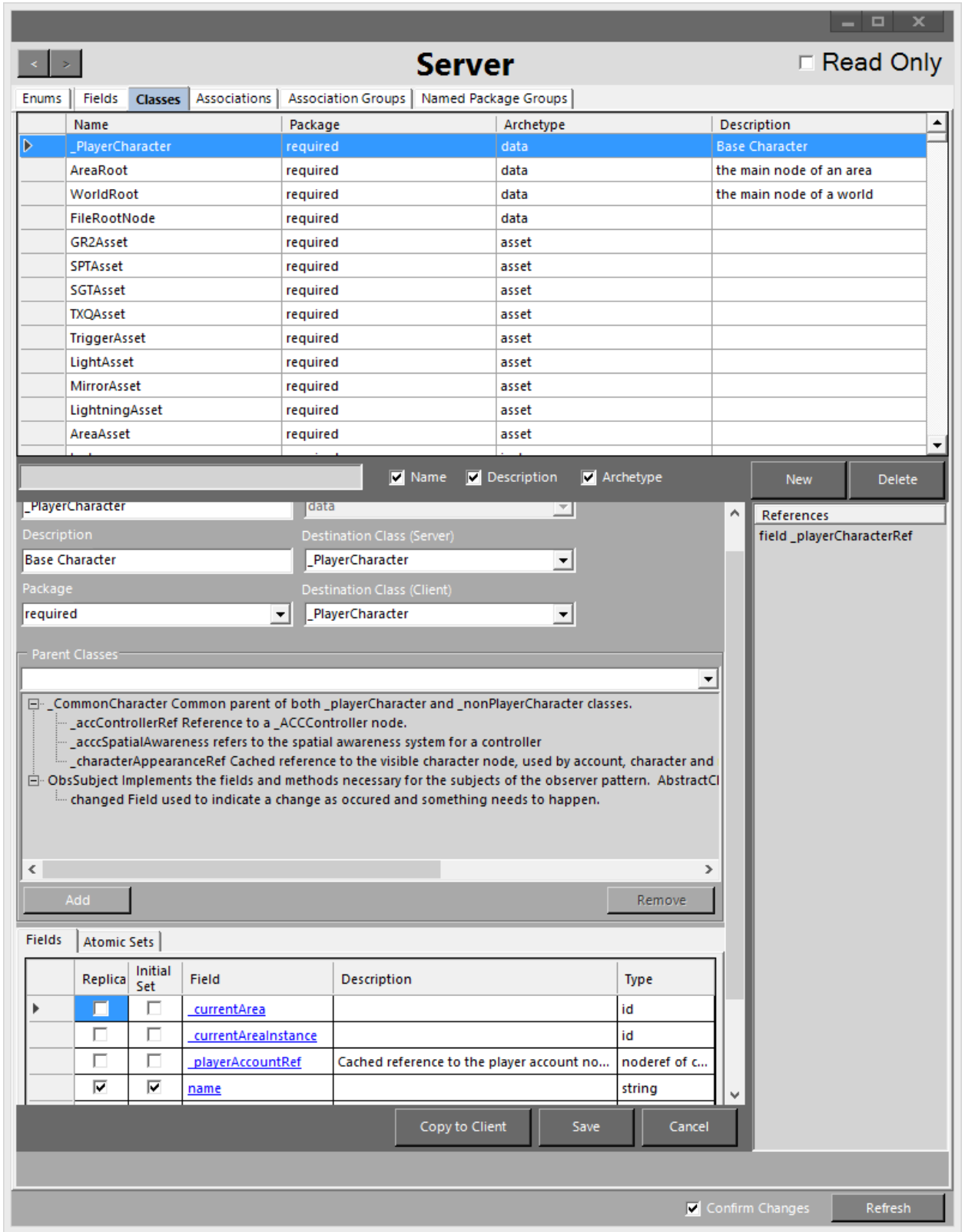
Introduction

In LSL, the data object model (classes, fields, enums, etc.) are defined independently of the scripting language (LSL). This is different than most computer languages, but gives LaniEngine it's distinctive power and real-time collaborative flexibility. The DOM can be manipulated with the CLI, but it is much easier to use the DOM Editor.

Overview

Using the DOM Editor, you may create, view, modify and delete fields, classes, enums, associations, association groups, and named package groups for both the client and the server. Changes made by the DOM editor or CLI are immediately available for use in script.


Client
Read Only
Enums
Fields
Classes
Associations
Association Groups
Named Package Groups
Name
Package
Description
enum_dockMode
required
docking mode
fxStates
required
List of valid states that an Fx can be in
fxEvents
required
List of valid events that occur for an Fx
fxStartMethods
required
List of methods which control how an FxAs...
fxStopMethods
required
List of methods which control how an FxAs...
FxRunStates
required
List of valid states an fxAssetController can...
fxTimerTypes
required
List of timer types for an Fx
InterpolationCurve
required
Used with the Interpolate function
TargetTypes
required
defines type of target
font
required
soundSizes
required
DraggableIconSources
required
Identifies what the source was when I pick...
DraggableIconTypes
required
Marks what kind of icon is being dragged
Name
Description
New
Delete
Name
enum_dockMode
Description
docking mode
Package
required
References
field dockMode
Values
Enter values, one per line
NONE
TOP
BOTTOM
LEFT
RIGHT
FILL
MARGINLESS_NONE
MARGINLESS_TOP
MARGINLESS_BOTTOM
MARGINLESS_LEFT
Named Package Groups
Name
Description
Copy to Server
Save
Cancel
Confirm Changes
Refresh



The Client DOM Editor and the Server DOM Editor are displayed independently, and available as separate LaniBlade options under the LaniScript menu. This allows maximum flexibility when you need to reference or

compare data in each. Each DOM datatype is displayed on a separately tabbed page. This layout allows users to browse and modify both the client and server datatypes independently.

By default, both the client and server DOMs are set to read-only mode. This can be independently enabled or disabled using the check box in the upper right of each DOM.

When a datatype entry is selected, its detail information is displayed below it in the lower portion of the UI.

Read Only Mode

By default, the DOM Editor opens in read-only mode to prevent accidental modifications of DOM definitions. In order to make any change to the DOM definitions, you must first clear the read-only flag located in the top right section of the DOM editor.

Confirm Changes

If the DOM editor is not in read-only mode, it defaults to prompt the user before it sends a change that can cause DOM modifications. When confirmation is active, a dialog box will pop up displaying the commands which it is about to send to the server, so they can be inspected for errors or unwanted changes. Clearing this checkbox causes the editor to pass all commands directly to the server without prompting for verification.

Datatype Definition Options

- New: Opens a dialog box which can be used to create a new datatype definition.
- Delete: Deletes the selected datatype definition.
- Refresh: Forces a query of the server for the list of all definitions of the currently active type.
- Glom: (Class Only) Finds the node currently selected in the viewport and executes the commands necessary to glom the current class onto the selected node.

The text box, left center, below the list of datatype definitions, allows you to search the list of definitions for specific entries. The left and right arrow buttons enable you to move to the previous and next definitions, respectively. The Name, Description, and, optionally Type and Archetype checkboxes, depending on the datatype context, allow you to restrict the search to a specific definition column.

Definition Detail Display and Editing

The lower portion of the UI displays detail information for the currently selected datatype definition. When not in read-only mode, changes may be made and saved to the definition of the datatype. Please refer to each datatype for more information on the specific type of information that can be modified.

- Enum: Refer to MED
- Field: Refer to MFD
- Class: Refer to MCD
- Association: Refer to MAD
- Association Group: Refer to MAG
- References: For datatypes which may be used by other classes and fields, this area lists which classes or fields make use of this definition. A definition cannot be deleted if it used by other definitions. Double clicking on a reference will take you to the UI tab and detail display for the reference; it will not modify the contents of the referencing tab.
- Atomic Sets (Class Only): Classes which have replicated fields may have atomic sets defined for their replicated fields. See also: Replication

Options

- Copy to Server / Copy to Client: Client DOM Editor option that will copy the current datatype definition information and reproduce the datatype on the server. The server button will do the same, transferring a copy of the data definition to the client
- Cancel: Abandons the changes made to the current definition.
- Save: Saves the changes made to the current definition to the server.